CHARUSAT
CHAROTAR UNIVERSITY OF SCIENCE AND TECHNOLOGY

CE251 : JAVA PROGRAMMING

Java Multithreading

Ms. Sachi Joshi
Assistant Professor

DEPSTAR

Devang Patel Institute of Advance Technology and Research

What are Multithreading Applications?

What is Multithreading?

- A multithreading program contains two or more parts that can run concurrently, Each part of such a program is called a **thread**, and each thread defines a separate path of execution.

A multithreading is a specialized form of multitasking.

Means execute more then one task at a time called multitasking

Difference b/w Process & Thread

- A process consists of the memory space allocated by the operating system that can contain one or more thread.
- A thread can't exit on its own, it must be a part of process.
- A process remains running until all of the non deamon threads are done executing.

So Multithreading

- Execute more than one thread at a time.
- Each & every thread is separate independent part of same application.

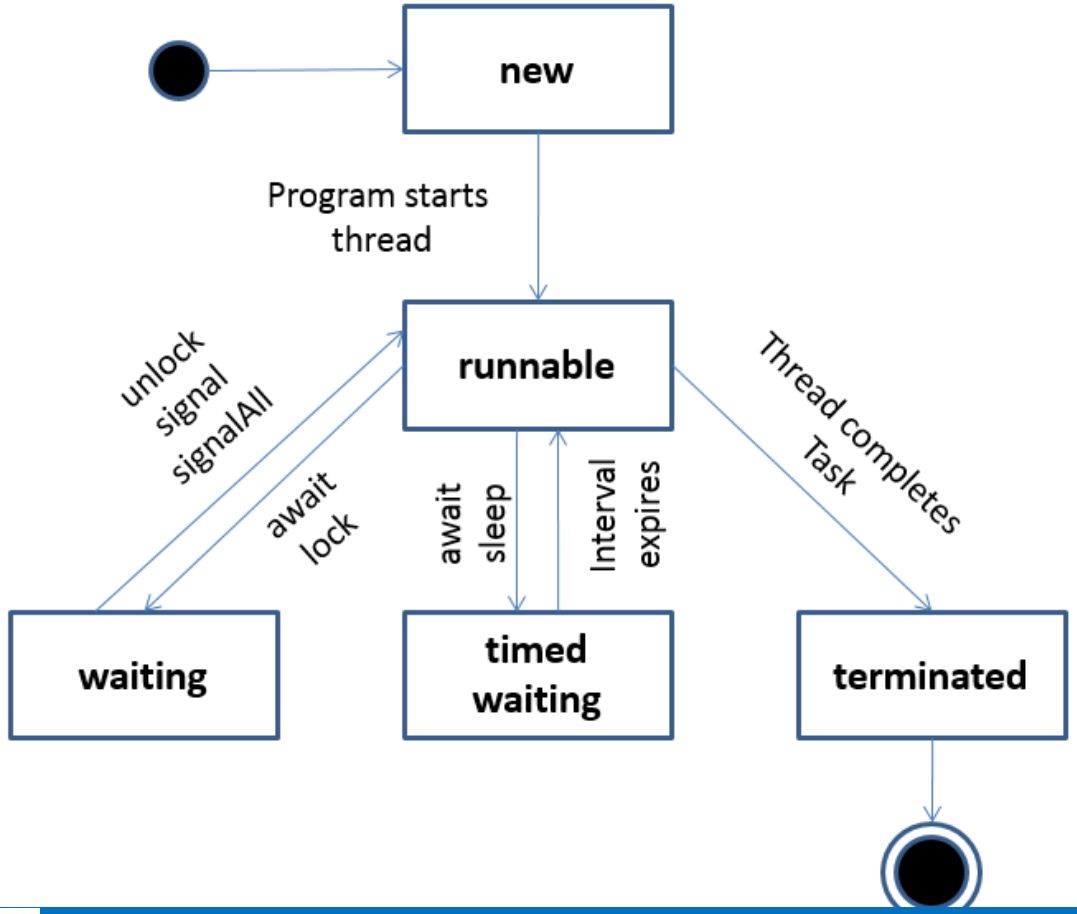
Life cycle of a Thread

- A thread goes through various stage in its life cycle.
- For example- a thread is born, started, runs & then dies

- There are 5 stages in the life cycle of the thread

1. New
2. Runnable
3. Waiting
4. Timed waiting
5. Terminated

Life cycle of a Thread (Cont.)



Creating a Thread in Java

- There are two ways in which we can create a Thread in Java

1. Creating a Thread by **extending** the **Thread class**
2. Creating a Thread by **implementing** the **Runnable interface**

Thread class & Runnable interface present in java.lang package

We will look in to each of this method with example

1) By Extending Thread Class

Step-1 create a new class that **extends Thread** class

Step-2 extending class must override the **run()** method, which is the entry point for the new thread.

Step-3 create an **instance of Thread** class

Step-4 call **start()** method to begin execution of the new thread.

Example

```
class MyThread extends Thread{
        public void run(){
       }
   }

MyThread t = new MyThread();

t.start();
```

Example

```
class MyThread extends Thread
{
  public void run()
  {
    for(int i=0; i<10; i++)
    {
   System.out.println("user thread");
    }
  }
}
```

```
class ThreadDemo
{
        public static void main(String[] args) {
                MyThread t = new MyThread();
                t.start();
          for(int i=0; i<10; i++)
          {
           System.out.println("main thread");
          }
        }
}
```

How many thread will be created?

2 thread

main thread created first

main thread will create another thread

Every thread have separate **stack memory**

t.start()

- At this line, application contain two thread
- main thread & user thread
- Which thread will execute first?

- Thread execution decided by **Thread scheduler**, one of the component of JVM

What happen during the execution?

- Where the start() method define?

- JVM will found start() method in **Thread** class

- Thread class start() method perform two action

1. Thread is registered to thread scheduler then thread will be created
2. Thread class start() method automatically call **run()** method

Output

```
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
user thread
user thread
user thread
user thread
user thread
user thread
user thread
user thread
user thread
user thread
user thread
```

Example-2 create another thread

```
class MyThread extends Thread
{
  public void run()
  {
    for(int i=0; i<4; i++)
    {
   System.out.println("user thread");
    }
  }
}
```

```
class ThreadDemo
{
        public static void main(String[] args) {
                MyThread t1 = new MyThread();
                t1.start();
                        MyThread t2= new
MyThread();
                t2.start();

          for(int i=0; i<4; i++)
          {
           System.out.println("main thread");
          }
        }
}
```

How many thread will be created?

3 thread

main thread created first

main thread will create **2 more thread**

Every thread have separate **stack memory**

This is the example that multiple thread performing single task

Output

```
user thread
user thread
user thread
user thread
main thread
main thread
main thread
main thread
user thread
user thread
user thread
user thread
```

OR

```
main thread
main thread
main thread
main thread
user thread
user thread
user thread
user thread
user thread
user thread
user thread
user thread
```

Example-3

```
class MyThread extends Thread
{
  public void run()
  {
    for(int i=0; i<10; i++)
    {
   System.out.println("call No = "+i);
          try{
                sleep(1000);
          }catch(InterruptedException
e){
               e.printStackTrace();
        }
    }}}
```

```
class ThreadDemo
{
        public static void main(String[] args) {
                MyThread t = new MyThread();
                t.start();
          for(int i=0; i<10; i++)
          {
           System.out.println("main thread");
          }
        }
}
```

Output

```
call No = 0
main thread
main thread
main thread
main thread
call No = 1
call No = 2
call No = 3
call No = 4
call No = 5
call No = 6
call No = 7
call No = 8
call No = 9
```

Example-4

```
class MyThread extends Thread
{
  public void run()
  {
    for(int i=0; i<5; i++)
    {
   System.out.println("call No = "+i);
          try{
                sleep(1000);
          }catch(InterruptedException
e){
               e.printStackTrace();
        }
    }}}
```

```
class ThreadDemo
{
        public static void main(String[] args) {
                MyThread t1 = new MyThread();
                t1.start();
                  MyThread t2 = new
MyThread();
                t2.start();
              for(int i=0; i<4; i++)
            {
             System.out.println("main thread");
            }
          }
}
```

Output

```
main thread
main thread
main thread
main thread
call No = 0
call No = 0
call No = 1
call No = 1
call No = 2
call No = 2
call No = 3
call No = 3
call No = 4
call No = 4
```

OR

```
call No = 0
main thread
main thread
main thread
main thread
call No = 0
call No = 1
call No = 1
call No = 2
call No = 2
call No = 3
call No = 3
call No = 4
call No = 4
```

Example- MyThread class without run() method

```
class MyThread extends Thread
{
}
```

Will it compile or not?

```
class ThreadDemo
{
        public static void main(String[] args) {
                MyThread t = new MyThread();
                t.start();
         for(int i=0; i<4; i++)
          {
           System.out.println("main thread");
          }
        }
}
```

Output-

```
D:\Java_2018\Multithreading>javac ThreadDemo.java

D:\Java_2018\Multithreading>java ThreadDemo
main thread
main thread
main thread
main thread
```

Thread class run() method will call with empty implementation- not recommended

Can we override the start() method?

```
class MyThread extends Thread
{
  public void start()
  {
    for(int i=0; i<4; i++)
    {
   System.out.println("call No = "+i);
         try{
               sleep(1000);
         }catch(InterruptedException e){
              e.printStackTrace();
         }    }
  }
}
```

```
class ThreadDemo
{
         public static void main(String[] args) {
                  MyThread t1 = new MyThread();
                  t1.start();
                           MyThread t2 = new
MyThread();
                  t2.start();

            for(int i=0; i<4; i++)
            {
             System.out.println("main thread");
            }
          }
}
```

Output-

```
D:\Java_2018\Multithreading>javac ThreadDemo.java

D:\Java_2018\Multithreading>java ThreadDemo
call No = 0
call No = 1
call No = 2
call No = 3
call No = 0
call No = 1
call No = 2
call No = 3
main thread
main thread
main thread
main thread
```

Overriding start() method but Thread will not created – means Thread class will not used

Can we overload run() method?

```
class MyThread extends Thread
{
  public void run()
  {
    for(int i=0; i<10; i++)
    {
   System.out.println("0-argument user thread");
    }
  }
public void run(int a)
  {
    for(int i=0; i<4; i++)
    {
   System.out.println("1-argument user thread");
    } }}
```

```
class ThreadDemo
{
        public static void main(String[] args) {

                MyThread t = new MyThread();
               t.start();

          for(int i=0; i<10; i++)
          {
           System.out.println("main thread");
          }
        }
}
```

Output-

```
D:\Java_2018\Multithreading>java ThreadDemo
main thread
main thread
main thread
main thread
main thread
main thread
0-argument user thread
0-argument user thread
0-argument user thread
0-argument user thread
0-argument user thread
0-argument user thread
0-argument user thread
0-argument user thread
0-argument user thread
0-argument user thread
main thread
main thread
main thread
main thread
```

We can overload run() method but JVM always call 0-argument run() method

How to call overloaded run(int a) method?

```
public void run()
  {
    for(int i=0; i<10; i++)
    {
   System.out.println("0-argument user thread");
    }
    run(100);
  }
```

Output-

```
D:\Java_2018\Multithreading>java ThreadDemo
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
0-argument user thread
0-argument user thread
0-argument user thread
0-argument user thread
0-argument user thread
0-argument user thread
0-argument user thread
0-argument user thread
0-argument user thread
0-argument user thread
1-argument user thread
1-argument user thread
1-argument user thread
1-argument user thread
```

How to write multiple functionality in run() method?

```
class MyThread extends Thread
{
  public void run()
  {       method1();
          method2();
          method3();
  }
  void method1(){      S.O.P("method1 called");
}
   void method2(){     SO.P("method2 called");
}
   void method3(){     S.O.P.("method3 called");
}
}
```

```
class RunFunctionDemo
{
  public static void main(String[] args)
    {
     MyThread t = new MyThread();
     t.start();
     }
}
```

Output-

```
D:\Java_2018\Multithreading>java RunFunctionDemo
method1 called
method2 called
method3 called
```

Create different thread & different task

```
class MyThreadOne extends Thread
{
  public void run()
  {       System.out.println("Thread one");   }
}
class MyThreadTwo extends Thread
{
  public void run()
  {       System.out.println("Thread two");
}
}
class MyThreadThree extends Thread
{
  public void run()
  {       System.out.println("Thread three");
}
}
```

```
class MultiThreadDemo
{
  public static void main(String[]
args)              {
     new MyThreadOne().start();
     new MyThreadTwo().start();
     new MyThreadThree().start();

  }
}
```

How many thread created here?

4

Thread Methods

1. setName(String)
2. getName()
3. CurrentThread()
4. getId()
5. isAlive()
6. activeCount()
7. setPriority(int)
8. getPriority()
9. setDaemon(boolean)
10. isDaemon()
11. join()
12. join(long)
13. interrupt()

Method Example

```
class MyThreadFour extends Thread
{
  public void run()
  {
    System.out.println("Thread Four");
  }
}

class ThreadMethods
{
  public static void main(String[] args) {
    MyThreadFour t1 = new MyThreadFour();
    t1.start();
    MyThreadFour t2 = new MyThreadFour();
    t2.start();
    System.out.println(t1.getName());
    System.out.println(t2.getName());

    t1.setName("mohammed");
    System.out.println(t1.getName());

    System.out.println(Thread.currentThread().getName());
    Thread.currentThread().setName("CHARUSAT");
    System.out.println(Thread.currentThread().getName());

    }
}
```

Output-

```
D:\Java_2018\Multithreading>java ThreadMethods
Thread Four
Thread Four
Thread-0
Thread-1
mohammed
main
CHARUSAT
```

Daemon Thread

- Background thread
- Thread which is working on background called daemon thread.
- Daemon thread provide support to foreground thread
- Low priority thread
- Its life depend on user thread

How to make Thread as Daemon Thread?

```
class MyThreadD extends Thread
{
  public void run()
  {
    System.out.println("Daemon Thread ");
  }
}

class DaemonThread
{
  public static void main(String[] args) {
    MyThreadD t1 = new MyThreadD();
    t1.setDaemon(true);
    t1.start();
    for(int i =0; i<10;i++)
    {
      System.out.println("main Thread ");
    }
}
}
```

Few points about daemon thread

- Once main thread is completed, daemon thread is terminated whether its completed or not.
- main thread never wait, daemon thread

Thread Priority

- Each thread have a priority.
- Priorities are represented by number b/w 1 and 10.
- Thread scheduler will use priority to schedule the threads.
- 3 constants fields are defined in Thread class:

1. public static int MIN_PRIORITY --→ default is 1
2. public static int NORM_PRIORITY --→ default is 5
3. public static int MAX_PRIORITY --→ default is 10

Example

```
public class ThreadPriority extends Thread
{
    public ThreadPriority(String tName)
    {
        super(tName);
    }
  public void run()
    {
        for(int i =0; i<10; i++)
        {
            try{
                Thread.sleep(1000);
                System.out.println("Call of " +this.getName() + i);
            }catch(InterruptedException e){
                e.printStackTrace();
            }
        }}
```

```
public static void main(String[] args) {
    ThreadPriority p = new ThreadPriority("Low  ");
    p.setPriority(Thread.MIN_PRIORITY);

    ThreadPriority p1 = new ThreadPriority("High ");
    p1.setPriority(Thread.MAX_PRIORITY);

    p.start();
    p1.start();
    for(int i =0; i<10; i++)
    {
        try{
            Thread.sleep(1000);
            System.out.println("Master thread " +i);
        }catch(InterruptedException e){
            e.printStackTrace();
        }
    }}}
```

Output

```
D:\Java_2018\Multithreading>java ThreadPriority
Call of High 0
Call of Low  0
Master thread 0
Call of High 1
Master thread 1
Call of Low  1
Call of High 2
Master thread 2
Call of Low  2
Call of High 3
Call of Low  3
Master thread 3
Call of High 4
Master thread 4
Call of Low  4
Call of High 5
Call of Low  5
Master thread 5
Call of High 6
Call of Low  6
Master thread 6
Call of High 7
Master thread 7
Call of Low  7
Master thread 8
Call of High 8
Call of Low  8
Call of High 9
Call of Low  9
Master thread 9
```

2) By Implementing Runnable Interface

- The easiest way to create a thread is to create a class that implements the Runnable interface.
- To implement Runnable, a class need to implement only a single method called run().
- After implementing Runnable, create **an object of class**
- Call the Thread class constructor **Thread(Runnable r, String name)**

Check Runnable Interface in lang package

```
D:\Java_2018\Exception>javap java.lang.Runnable
Compiled from "Runnable.java"
public interface java.lang.Runnable {
  public abstract void run();
}
```

Commonly used Constructors of Thread class:

- Thread()
- Thread(String name)
- Thread(Runnable r)
- Thread(Runnable r, String name)

Example

```
class RunnableThread implements Runnable{
public void run(){
        System.out.println("thread is running...");
}public static void main(String args[]){
        RunnableThread m1=new RunnableThread();
        Thread t1 =new Thread(m1);
        t1.start();
  }
 }
```

Example-2

```
class MyRunnable implements Runnable
{

    public void run()
    {
        for(int i =0; i<10; i++)
        {
            try{
                Thread.sleep(1000);
                System.out.println("Child thread " +i);
            }catch(InterruptedException e){
                e.printStackTrace();
            }
        }
    }
}
```

```
public class MyRunner{
    public static void main(String[] args) {

        MyRunnable m = new MyRunnable();
        Thread t = new Thread(m);
        t.start();
        for(int i =0; i<10; i++)
        {
            try{
                Thread.sleep(1000);
                System.out.println("Master thread " +i);
            }catch(InterruptedException e){
                e.printStackTrace();
            }
        }
    }
}
```

Output

```
D:\Java_2018\Multithreading\Runnable>java MyRunner
Master thread 0
Child thread 0
Master thread 1
Child thread 1
Master thread 2
Child thread 2
Master thread 3
Child thread 3
Master thread 4
Child thread 4
Child thread 5
Master thread 5
Master thread 6
Child thread 6
Child thread 7
Master thread 7
Master thread 8
Child thread 8
Master thread 9
Child thread 9
```

```
D:\Java_2018\Multithreading\Runnable>java MyRunner
Child thread 0
Master thread 0
Master thread 1
Child thread 1
Master thread 2
Child thread 2
Master thread 3
Child thread 3
Master thread 4
Child thread 4
Master thread 5
Child thread 5
Master thread 6
Child thread 6
Master thread 7
Child thread 7
Master thread 8
Child thread 8
Child thread 9
Master thread 9
```

Example-3

```
class MyRunnable implements Runnable
{
    Thread t;
    public MyRunnable()
    {
        t = new Thread(this, "My Threa
        System.out.println("Child thre
        t.start();
    }
    public void run()
    {
        for(int i =0; i<10; i++)
        {
            try{
                Thread.sleep(1000);
                System.out.println("Child thread " +i);
            }catch(InterruptedException e){
                e.printStackTrace();
            }
        }
    }
}
```

```
public class MyRunner{
    public static void main(String[] args) {
        new MyRunnable();
        for(int i =0; i<10; i++)
        {
            try{
                Thread.sleep(1000);
                System.out.println("Master thread " +i);
            }catch(InterruptedException e){
                e.printStackTrace();
            }
        }
    }
}
```

Output-

```
D:\Java_2018\Multithreading\Runnable>java MyRunner
Child thread Thread[My Thread,5,main]
Child thread 0
Master thread 0
Master thread 1
Child thread 1
Master thread 2
Child thread 2
Master thread 3
Child thread 3
Master thread 4
Child thread 4
Master thread 5
Child thread 5
Master thread 6
Child thread 6
Master thread 7
Child thread 7
Child thread 8
Master thread 8
Child thread 9
Master thread 9
```

```
D:\Java_2018\Multithreading\Runnable>java MyRunner
Child thread Thread[My Thread,5,main]
Master thread 0
Child thread 0
Master thread 1
Child thread 1
Master thread 2
Child thread 2
Master thread 3
Child thread 3
Master thread 4
Child thread 4
Master thread 5
Child thread 5
Child thread 6
Master thread 6
Master thread 7
Child thread 7
Master thread 8
Child thread 8
Child thread 9
Master thread 9
```

How to prevent Thread Execution?

- Thread offer 3 methods to prevent Thread execution

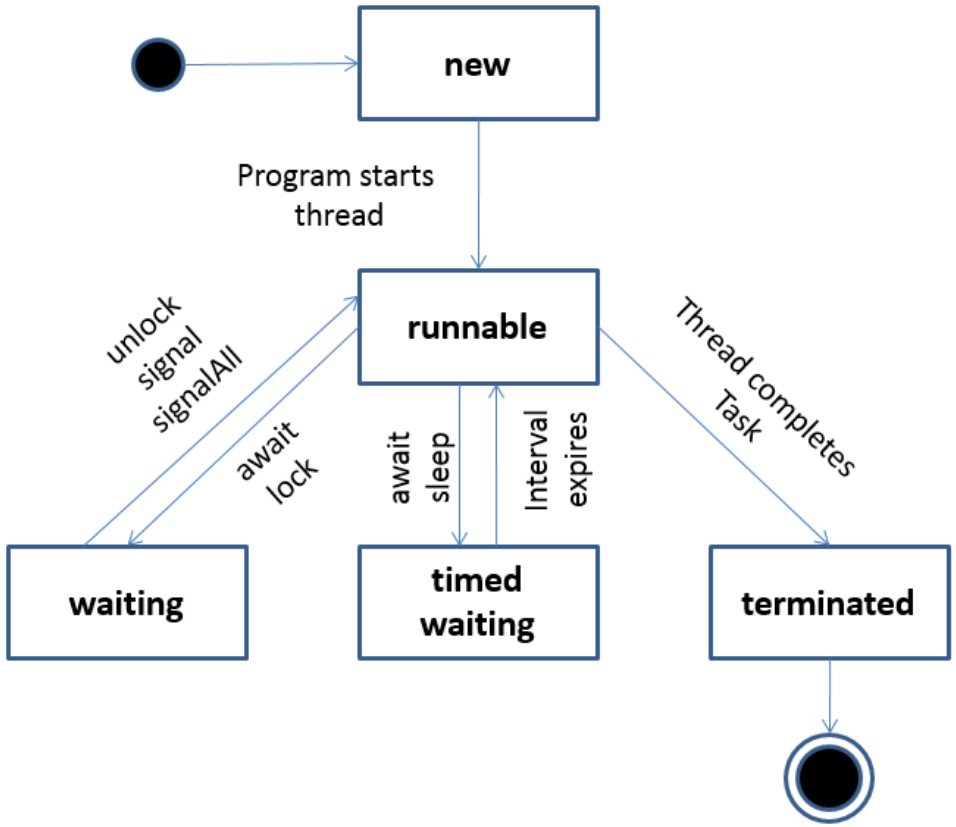
1. yield()
2. join()
3. sleep()

What is the yield() method?

- Let me tell u story – Mobile phone & BSNL landline phone

- So yield() method pauses current execution to give the chance for remaining waiting thread of same priority.
- If no other thread is waiting or all waiting threads have low priority then same thread can continue its execution.

yield() method in thread life cycle



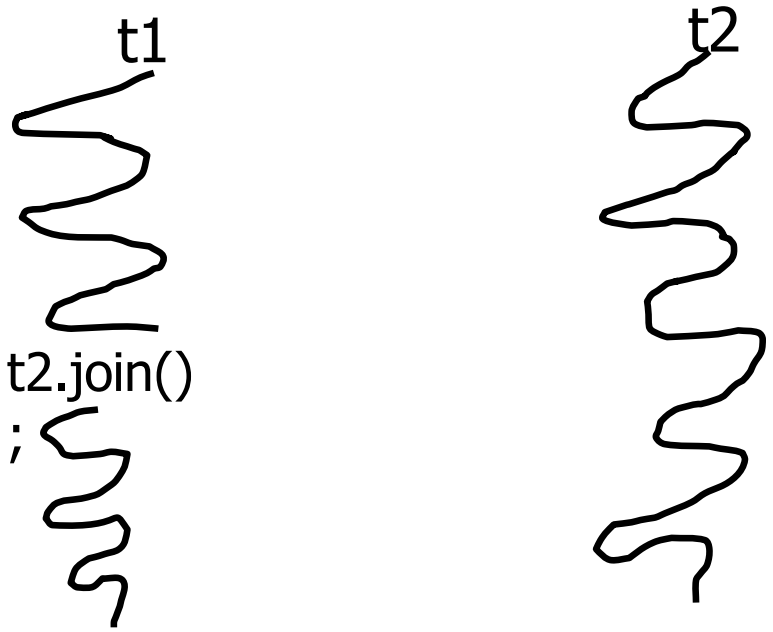
Example

```
class MyThreadYield extends Thread
{
    public void run()
    {
      for(int i=0; i<10; i++)
      {
       System.out.println("child thread");
       Thread.yield();
      }
    }
}
```

```
class ThreadYieldDemo
{
    public static void main(String[] args) {
        MyThreadYield t = new MyThreadYield();
        t.start();
      for(int i=0; i<10; i++)
      {
       System.out.println("user thread");
      }
    }
}
```

join() method

- If a Thread want to wait until the completion of other thread then join() method must call.



3 types of join() methods in Thread class

1. public final void join() throws InterruptedException
2. public final void join(long) throws InterruptedException
3. public final void join(long, int) throws InterruptedException

Example

```
class MyThreadJoin extends Thread
{
    public void run()
    {
      for(int i=0; i<10; i++)
      {
       System.out.println("child thread");
       try{
        Thread.sleep(2000);
       }
       catch(InterruptedException ie){
        ie.printStackTrace();
       }
      }
    }
}
```

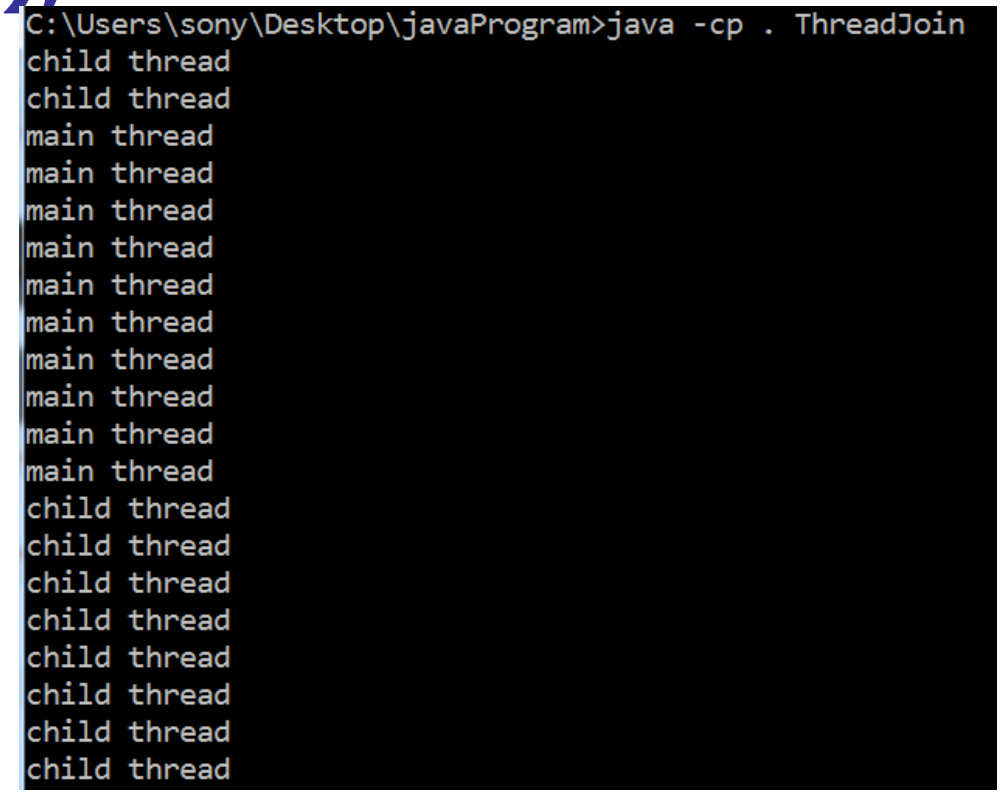
```
class ThreadJoin
{
    public static void main(String[] args) throws InterruptedException{
        MyThreadJoin t = new MyThreadJoin();
        t.start();
        t.join();
      for(int i=0; i<10; i++)
      {
       System.out.println("main thread");
      }
    }
}
```

Output

```
C:\Users\sony\Desktop\javaProgram>java -cp . ThreadJoin
child thread
child thread
child thread
child thread
child thread
child thread
child thread
child thread
child thread
child thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
```

Small change in join() method-

t.join(4000);



Another Small change in join() method-

t.join(4000, 2000);

```
C:\Users\sony\Desktop\javaProgram>java -cp . ThreadJoin
child thread
child thread
main thread
main thread
child thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
child thread
child thread
child thread
child thread
child thread
child thread
child thread
```

Most valuable concept in multithreading is Synchronization

- People know multithreading but not good in synchronization
- Little bit difficult

Few basic point about **synchronized** modifier

- Class can't be **synchronized**
- Variables can't be **synchronized**

- Method can be **synchronized**
- Blocks can be **synchronized**

What is the purpose of synchronization?

- **For example-** if multiple threads try to write within a same file then they may corrupt the data because one of the threads can override data or while one thread is opening the same file at the same time another thread might be closing the same file.

- If resources are shared
- Railway Tatkal reservation – vary good example
- Bank transaction using ATM, Online etc.

Synchronization in Java

- Synchronization in java is the capability *to control the access of multiple threads to any shared resource*.
- Java Synchronization is better option where we want to allow only one thread to access the shared resource.
- The synchronization is mainly used to
- To prevent thread interference.
- To prevent data inconsistency problem.

Example

```
class DisplayMethods
{
  void show(String name)
  {
   System.out.println("M1 sync method");
   for(int i=1;i<=5;i++)
   {
      System.out.print("Good Morning: ");
     try{
        Thread.sleep(400);
      }
     catch(Exception e){System.out.println(e);}
     System.out.println(name);
     }

   }
}
```

```
class FirstThread extends Thread
{
  DisplayMethods d;
  String name;
  FirstThread(DisplayMethods d, String name)
  {
    this.d=d;
    this.name = name;
  }
  public void run()
  {
    d.show(name);
  }
}
```

```
class SynchronizedExam
{
  public static void main(String[] args) {
    DisplayMethods d = new DisplayMethods();
    FirstThread t1 = new FirstThread(d, "mohammed");
    FirstThread t2 = new FirstThread(d, "bohara");

    t1.start();
    t2.start();
  }
}
```

Output

```
C:\Users\sony\Desktop\javaProgram>javac SynchronizedExam.java

C:\Users\sony\Desktop\javaProgram>java -cp . SynchronizedExam
M1 sync method
Good Morning: M1 sync method
Good Morning: mohammed
Good Morning: bohara
Good Morning: mohammed
Good Morning: bohara
Good Morning: mohammed
Good Morning: bohara
Good Morning: mohammed
Good Morning: bohara
Good Morning: mohammed
bohara
```

Irregular outcome

Now add synchronized modifier with method

synchronized void show(String name){ }

Output

```
C:\Users\sony\Desktop\javaProgram>java -cp . SynchronizedExam
M1 sync method
Good Morning: mohammed
Good Morning: mohammed
Good Morning: mohammed
Good Morning: mohammed
Good Morning: mohammed
M1 sync method
Good Morning: bohara
Good Morning: bohara
Good Morning: bohara
Good Morning: bohara
Good Morning: bohara
```

regular outcome

Lets do small modification in previous program

Create two object of DisplayMethod and assign it to different object and check the outcome.

```
class SynchronizedExam
{
  public static void main(String[] args) {
    DisplayMethods d1 = new DisplayMethods();
    DisplayMethods d2 = new DisplayMethods();
    FirstThread t1 = new FirstThread(d1, "mohammed");
    FirstThread t2 = new FirstThread(d2, "bohara");

    t1.start();
    t2.start();
  }
}
```

Output

```
C:\Users\sony\Desktop\javaProgram>javac SynchronizedExam.java

C:\Users\sony\Desktop\javaProgram>java -cp . SynchronizedExam
M1 sync method
Good Morning: M1 sync method
Good Morning: mohammed
Good Morning: bohara
Good Morning: mohammed
Good Morning: bohara
Good Morning: mohammed
Good Morning: bohara
Good Morning: mohammed
Good Morning: bohara
Good Morning: mohammed
bohara
```

Irregular outcome- even method is synchronized

One solution for such problem

- Make method static and synchronized both

synchronized static void show(String name)

Check the outcome again

Output

Called as class level lock

```
C:\Users\sony\Desktop\javaProgram>java -cp . SynchronizedExam
M1 sync method
Good Morning: mohammed
Good Morning: mohammed
Good Morning: mohammed
Good Morning: mohammed
Good Morning: mohammed
M1 sync method
Good Morning: bohara
Good Morning: bohara
Good Morning: bohara
Good Morning: bohara
Good Morning: bohara
```

regular outcome- bcs it will make class synchronized

Create two synchronized method and one normal method

```
class SyncMethods
{
    synchronized void m1(int n)
    {
     System.out.println("M1 sync method");
     for(int i=1;i<=5;i++)
     {
        System.out.println(n*i);
     try{
        Thread.sleep(400);
      }
     catch(Exception e){System.out.println(e);}
     }

    }
    synchronized void m2()
    {
     System.out.println("M2 sync method");
     }
     void m3()
    {
     System.out.println("M3 normal method");
     }
}
```

```java
class ThreadOne extends Thread
{
    SyncMethods s;
    ThreadOne(SyncMethods s)
    {
        this.s=s;
    }
    public void run()
    {
        s.m1(5);
    }
}
```

```java
class ThreadTwo extends Thread
{
    SyncMethods s;
    ThreadTwo(SyncMethods s)
    {
        this.s=s;
    }
    public void run()
    {
        s.m2();
    }
}
```

```java
class SynDemo
{
    public static void main(String[] args) {

        SyncMethods s = new SyncMethods();

        ThreadOne t1 = new ThreadOne(s);
        ThreadTwo t2 = new ThreadTwo(s);
        t1.start();
        t2.start();

    }
}
```

```
C:\Users\sony\Desktop\javaProgram>java -cp . SynDemo
M1 sync method
5
10
15
20
25
M2 sync method
```

Do one exercise

```
class DisplayMethods
{
  synchronized static void show(String name)
  {
   System.out.println("M1 sync method");
   for(int i=1;i<=5;i++)
   {
      System.out.print("Good Morning: ");
     try{
        Thread.sleep(400);
      }
     catch(Exception e){System.out.println(e);}
     System.out.println(name);
     }
  }
    synchronized static void m2()
    {
      System.out.println("M2 sync static method");
    }
    synchronized void m3()
    {
      System.out.println("M3 sync method only");
    }
    static void m4()
    {
      System.out.println("M4 static method only");
    }
    void m5()
    {
      System.out.println("M5 normal method");
    }
}
```

Is there any problem with synchronized method?

- Yes

Synchronized block

Three ways

1. Get the lock of current object
2. Get a lock of particular object ‘b’
3. Get a class level lock ‘DisplayMethods.class’

Get the lock of current object

```
synchronized(this)
{
}
```

Get a lock of particular object ‘b’

```
synchronized(b)
{
}
```

Get a class level lock 'DisplayMethods.class'

```
synchronized(DisplayMethods.class)
{
}
```

```
class MethodsWithSynBlock
{
  void show(String name)
  {
   System.out.println("M1 sync method");
   ;;;;;;;;;;;;;;;;;;;
   System.out.println("100th statement");
   ;;;;;;;;;;;;;;;;;;;
   System.out.println("1000th statement");
   ;;;;;;;;;;;;;;;;;;;
   synchronized(this)
   {
    for(int i=1;i<=5;i++)
    {
      System.out.println("Good Morning: "+name);
     try{
        Thread.sleep(400);
      }
     catch(Exception e){System.out.println(e);}
     }
   }
     ;;;;;;;;;;;;;;;;
     System.out.println("10000th statement");
     ;;;;;;;;;;;;;;;;
     System.out.println("15000th statement");
     ;;;;;;;;;;;;;;;;
  }
   void m5()
    {
      System.out.println("M5 normal method");
    }
}
```

```
class FirstThread extends Thread
{
  MethodsWithSynBlock d;
  String name;
  FirstThread(MethodsWithSynBlock d, String name)
  {
    this.d=d;
    this.name = name;
  }
  public void run()
  {
    d.show(name);
  }
}
```

```
class SynchronizedBlock
{
  public static void main(String[] args) {
    MethodsWithSynBlock d1 = new MethodsWithSynBlock();
    FirstThread t1 = new FirstThread(d1, "mohammed");
    FirstThread t2 = new FirstThread(d1, "bohara");

    t1.start();
    t2.start();
  }
}
```

Example

Output

```
C:\Users\sony\Desktop\javaProgram>java -cp . SynchronizedBlock
M1 sync method
M1 sync method
100th statement
1000th statement
100th statement
1000th statement
Good Morning: mohammed
Good Morning: mohammed
Good Morning: mohammed
Good Morning: mohammed
Good Morning: mohammed
Good Morning: bohara
10000th statement
15000th statement
Good Morning: bohara
Good Morning: bohara
Good Morning: bohara
Good Morning: bohara
10000th statement
15000th statement
```

Any Question

Q-1

```
class Job extends Thread {
    private Integer number = 0;
    public void run() {
        for (int i = 1; i < 1000000; i++) {
            number++;
        }
    }
    public Integer getNumber() {
        return number;
    }
}
public class Question1 {
    public static void main(String[] args){
        Job thread = new Job();
        thread.start();
        System.out.println(thread.getNumber());
    }
}
```

What is the output?

A. It prints 0.

B. It prints 999999.

C. The output is not guaranteed to be any of the above.

Q-2

```
class Job extends Thread {
    private Integer number = 0;
    public void run() {
        for (int i = 1; i < 1000000; i++) {
            number++;
        }
    }
    public Integer getNumber() {
        return number;
    }
}
public class Question1 {
    public static void main(String[] args) throws InterruptedException{
        Job thread = new Job();
        thread.start();
        thread.join();
        System.out.println(thread.getNumber());
    }
}
```

What is the output?

A. It prints 0.

B. It prints 999999.

C. The output is not guaranteed to be any of the above.

Q-3 What is the output?

```
public class Threads3 implements Runnable {
        public void run() {
                System.out.print("running");
        }
        public static void main(String[] args) {
                Thread t = new Thread(new
Threads3());
                t.run();
                t.run();
                t.start();
        }
}
```

A. Compilation fails.

B. An exception is thrown at runtime.

C. The code executes and prints "running".

D. The code executes and prints "runningrunning".

E. The code executes and prints "runningrunningrunning".

Q-4 **What is the output?**

```
public class TestOne implements Runnable
{
    public static void main (String[] args) throws Exception
    {
        Thread t = new Thread(new TestOne());
        t.start();
        System.out.print("Started");
        t.join();
        System.out.print("Complete");
    }
    public void run()
    {
        for (int i = 0; i < 4; i++)
         {
            System.out.print(i);
        }
    }
}
```

A. Compilation fails.

B. An exception is thrown at runtime.

C. The code executes and print "StartedComplete".

D. The code executes and print "StartedComplete0123".

E. The code executes and prints "Started0123Complete".

Q-5 **What is the output?**

```
class DifferentThread
{
    public static void main(String[] args) {

    Runnable r = new Runnable() {
        public void run()
        {
            System.out.print("Cat");
        }
    };

 Thread t = new Thread(r) {
        public void run()
        {
            System.out.print("Dog");
        }
    };

    t.start();
    }

}
```

A. Cat

B. Dog

C. Compilation fails.

D. The code runs with no output.

E. An exception is thrown at runtime.